

Weekly Report(2018.12.10-2018.12.16)

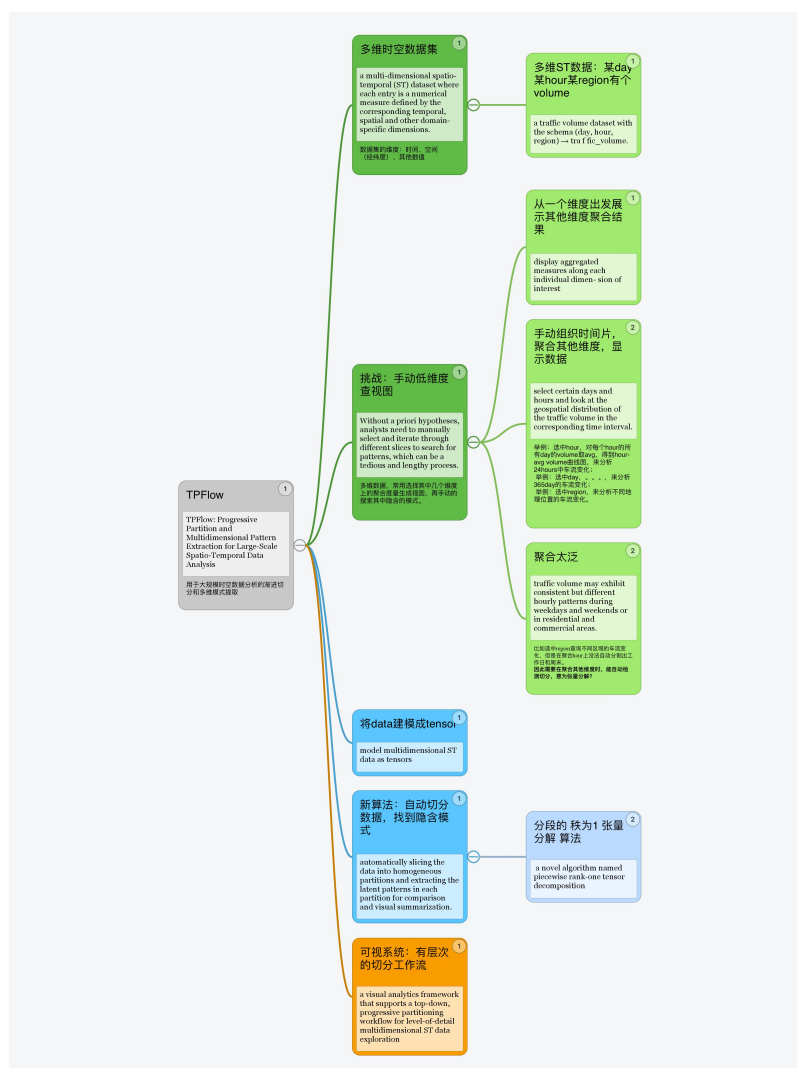
DONE

1. 有关TCPTree中的基于閾值的时间片截取算法研究:

- 看了英语原论文，中文论文，代码。简单的说，是对每一个1对1节点的关联关系（diff值）的时间序列进行了阈值切分，此后取并集得到1对多节点的时间片切分结果。
- 详细报告在附录中。

2. 看TPFlow论文和博客:

- 先前工作
 - 1. 大二聚类项目 掌握**C和C++**处理大数据文件、字符串；
 - 2. 大三数学建模比赛 掌握**MATLAB**字符串匹配、矩阵运算；
 - 3. 大三人工智能课程 掌握**Python**的SOM、SVM的实现。
- 当前任务
 - 1. 看论文TP FLOW原文和中文博客；
 - 2. 数据挖掘技巧总结；
 - 3. 张量学习。
- 简单报告



TPFlow思维导图

小结

- 本周，首先将TCPTree的阈值算法分析完成，然后开始了TPFlow的算法研究。

学习记录

学习日期	学习事项	学习时间
周一	看论文、代码，写算法报告	5h
周二	玉泉上课	5h
周三	整理	4h
周四	玉泉体检，上课	5h
周五	开组会，做实验	7h
周六	看TCPFlow	3h
周日	看TCPFlow	3h

PLAN

短期计划

- 继续看TPFlow。

中期计划

- 自学less、webpack、promise、\$.ajax、VS Code + Git等等。

长期计划

- 推进VIS 2019的项目进度。
- 使用所学的网页工具实现自己的idea。

APPENDIX

阈值算法详解

TCPTree 阈值算法 英语原文和图例

时间切分应用在 1 对多节点和 1 对 1 节点上。每个树节点都绑定了关联关系的时间序列。通过时间切分来获取显著相关的时间片，并扩展成新的树节点。

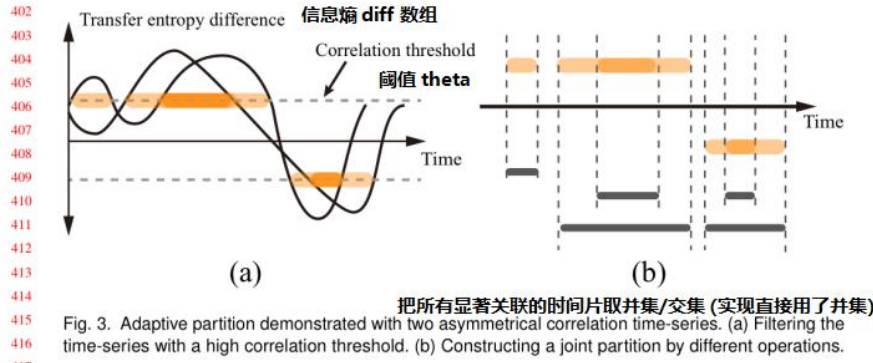
一种基于阈值的方法将预览用户感兴趣的切分方式。其可直接应用到 1 对 1 节点上。对于 1 对多节点，由于其中的每一个 1 对 1 节点关联关系各不相同，会产生各自不同的时间切分预览，我们的方法是：

- ①：先对每一个 1 对 1 节点执行阈值算法。
- ②：将第一步得到的不同的切分时间片合并起来，再生成新的时间节点。

389 Time partition is applied on one-to-many nodes and one-to-one nodes in variable tree. In a TCP
390 tree, each tree node is attached with a correlation time-series. By time partition, time pieces with
391 significant relevance are extracted, each of which is appended to a new tree node. A threshold-based
392
393 method is designed to obtain the user-interested partitions, e.g. the partitions have high $TD(X, Y)$ and
394 so on, as shown in Figure 3. For one-to-one nodes, the method is directed applied. For one-to-many

nodes, one critical problem is the conflict process of different partitions of all involved time-series, because each one-to-one correlation time-series can have independent partition scheme. We solve this problem with a two-stage process. In the first stage, the threshold-based method is applied to all correlation time-series in the node. In the second stage, we construct a joint partition by leveraging the obtained partitions in the first stage, and then build a time tree based on the partition.

In this way, the structure of TCPTree is dynamically built according to the partition operations.



图例说明：对于 2 个非对称的关联时间序列进行了自适应的切分。(a) 使用高的关联阈值对时间序列进行过滤。(b) 不同的操作生成 1 个联合切分。

TCPTree 阈值算法 Java 代码框架 (见阈值算法.java)

getLinkedPartition

- acc
 - getTEThresholdPartition
 - getMIThresholdPartition
- expand

TCP 阈值算法 伪代码解析

[htb] Algorithm 1

输入: 动态图数据的关系矩阵 Cor , 阈值 θ , 分割模式 $mode$

输出: 分割结果 $partitions$

```

1: timeSegmentArray = [], i = 0
2: for 对于  $Cor$  中的每一行  $c$  do cor 中的每一行就是 1 对 1 节点的关联关系矩阵
3:   timeSegments = [] line3~line16 代码段中:
4:   for  $c$  中的每个关系三元组  $v$  do baseValue 即为 diff[i];
5:      $t$  为  $v$  所在的时刻 timeSegments 记录所有时间片是否是显著关联关系 (True 或
6:      $T_{a \rightarrow b}, T_{b \rightarrow a}, I = v$  False 表示)
7:      $TD = T_{a \rightarrow b} - T_{b \rightarrow a}, TS = T_{a \rightarrow b} + T_{b \rightarrow a}$  参见函数 getTEThresholdPartition
8:      $baseValue = base(T_{a \rightarrow b}, T_{b \rightarrow a}, I, TD, TS, mode)$ 
9:     if  $|baseValue| > \theta$  then
10:      timeSegments[t] = True;
11:     else
12:      timeSegments[t] = False;
13:     end if
14:   end for
15:   timeSegmentArray[i] = timeSegments, i ++
16: end for

17: partition = [], t = 0, segment = [], index = 0, state = -1, sIndex = 0
18: while  $t > timestamps$  do line17~41 代码段中:
19:   if state == -1 then 实则是将所有 1 对 1 节点的显著关联关系的时间片取了并集
20:     for each  $ele$  in timeSegmentArray do 参见函数 getLinkedPartition 和 acc
21:       if  $ele[t]$  then
22:         state = 0, segment[sIndex] = t, sIndex ++
23:         break
24:       end if
25:     end for
26:   end if
27:   if state == 0 then
28:     flag = False
29:     for each  $ele$  in timeSegmentArray do
30:       flag = ele[t]
31:     end for
32:     if flag then
33:       segment[sIndex] = t, sIndex ++
34:     else
35:       partition[index] = segment, index ++

```

```
35:         per segment [process] = segment, remove = 1
36:         segment = [], sIndex = 0
37:         state = -1
38:     end if
39: end if
40:     t+ = 1;
41: end while
42: return 如果 qualified = TRUE 满足, 返回对应的四元组
```

阈值算法